IT UNIVERSITY OF COPENHAGEN

DevOps, Software Evolution and Software Maintenance

COURSE CODE: KSDSESM1KU

MSC IN COMPUTER SCIENCE

DevOps: ITU-MiniTwit

GROUP E — GRL PWR

IT UNIVERSITY OF COPENHAGEN

Name	Email
Amalie Bøgild Sørensen	abso@itu.dk
Andreas Nicolaj Tietgen	anti@itu.dk
Malin Birgitta Linnea Nordqvist	bino@itu.dk
Mille Mei Zhen Loo	milo@itu.dk
My Marie Nordal Jensen	myje@itu.dk
Sarah Cecilie Chytræus Christiansen	sacc@itu.dk

May 10, 2024

2500 words

Table of Contents

1 System Perspective

1.1 Current state

Table 1 describes the status of the system after the simulator has been shut down.

Total requests	34233
Requests secured	34233
Total unhandled exceptions	18
P99 min response time	94.2 ms
P99 max response time	716 ms

Table 1: *System metrics*

Furthermore, the static analysis tool Sonarcloud assesses that the quality of the overall code is good (see appendix B).

2 Process Perspective

hallo [2]

2.1 Monitoring and logging

For monitoring we use Grafana. Our board shows requests duration, errors rate, top 10 unhandled exception endpoints and more (for visualization of the board, see appendix A). We have used the board to get an overview of where to put our focus, ie. which endpoints to improve, which errors to fix, etc. Moreover, we have gained insights into the health of the system and gotten an impression of how the backend and frontend handle the requests.

For logging, we use Grafana and Loki. It seemed obvious to continue our work with Grafana in order to keep the system setup as simple as possible. The logs are divided into Information, Warning, Debug and Error. All logging statements are placed in the controllers, such that we have information about the users' whereabouts in the system. For example, we log when a user logins whether successful or unsuccessful.

2.2 Security Assessment

Based on our security assessment, which can be found in appendix C, we constructed the following risk matrix:

	Rare	Unlikely	Possible	Likely	Certain
Catastrophic	Exposed Secrets		Depricated Dependencies		Open Ports
Critical	SQL Injection	URL tampering	Password Brute Forcing		
Marginal				Log Injection	
Negligible					
Insignificant					

Figure 1: *Risk matrix from security assessment*

We decided to focus on the scenarios in the red zone, as they would be the cause of most damage. Due to the group not having enough knowledge about how the simulator worked in regards to creating users and logging in, we didn't move further with the *Password Brute Forcing* scenario, whereas the solution would have been to incorporate 2FA, password requirements and time-out limitation for an actual system in production.

For *Log Injection*, we made sure to sanitize user inputs, as they beforehand were put directly into the log, which could be exploited to hide ill-intentioned actions. We thereby hardened the system against injection attacks. With *Depricated Dependencies*, we chose to integrate the tool *dependabot* into our CI/CD pipeline to watch out for outdated dependencies. This helps us ensure that an adversary isn't able to take advantage of the team not being aware of vulnerabilities in variuos of the used tools.

Open Ports was deemed the biggest risk, as this was most likely the reason for us being hacked, which we have written more about in our lesson about being hacked. We used the tool *nmap* to check for open ports, where we then tried to close the open port used for Prometheus. It was unclear if we were successful as *nmap* kept showing the port as open despite seeing the firewall configuration saying otherwise on the server itself. We also attempted to switch from HTTP to HTTPS, as it would strengthen the overall security of our application, but here we ran into several bureaucratic issues with changing the name servers for the domain, which resulted in us running out of time of the task. Ideally, this would have been accomplished to further harden our system's security.

3 Lessons Learned

3.1 Lesson 1: Getting Hacked

Just a couple of hours after attending the lecture on security, we got hacked, leading us to experience firsthand how important it is to incorporate security in a CI/CD pipeline.

The first suspicion we got was when we discovered, through our monitoring, that the response time of our server was suddenly very slow. This led us to our Digital Ocean dashboard which showed that the server was using 100% CPU power, which is highly unusual.

From that point the group scoured the server for clues as to what was happening, finding countless calls to `masscan` essentially drowning our server, as well as mysterious installations and what looked like a call to a remote script via a cronjob.

After a few failed attempts to evict the adversary it was decided to destroy the server, as we fortunately had already implemented our Infrastructure as Code, so provisioning and deploying a new server could be done in under half an hour.

After some introspection into our system, we assume that the adversary had gotten access via some ports that we were unaware were exposed. The ports became exposed in an attempt to make the network function between servers in a docker swarm, which seems to override the firewall.

Learning from this, we have worked to close exposed ports from Docker and finding alternative solutions to setting up the network. Another key takeaway is that because we had the necessary monitoring in place, to figure out that the server was being targeted, as well as having our Infrastructure as Code, we were able to detect and react to the attack fairly quickly, giving us only a few hours of downtime.

3.2 Lesson 2: Shift from Vagrant to Ansible-Pulumi

Another lesson we have learned during this project is the importance of choosing the right tools for a project. At the start of the project, we had chosen to provision our VMs with Vagrant, inspired by the exercises from the course. When realizing later in the project that we would have to switch Digital Ocean account at some point due to running out of credit, we had to streamline the setup of our VMs. The choice of Configuration Management tool fell upon Ansible, which was supposed to call the Vagrant files from a config server, provisioning the web and monitoring servers. However, when having more complicated automation and collaboration needs for our project, it turned out that Vagrant was not the right tool for the job. After many hours of attempting to get Vagrant to work with Ansible, we found out that Vagrant saves local metadata to maintain some state, which was making the provisioning from Ansible and the config server fail[1]. Furthermore, Vagrant does not seem to be supported with GitHub Actions - possibly also due to how it handles state. We decided to cut our losses with Vagrant and search for a more suitable tool that could help us write our infrastructure as code. The choice fell upon Pulumi in the end. It was a valuable lesson to see how it made a difference when we took the time to investigate different tools and their properties so we could make an informed decision based on the knowledge of our system's needs.

4 References

[1] DevOps Group e. *Github Issue: Add playbooks in the vagrant provisioning steps*. 2024. URL: https://github.com/devops2024-group-e/itu-minitwit/issues/178.

[2] Gene Kim. *The DevOPS Handbook - How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. It Revolution Press, 2016. ISBN: 9781942788003.

[3] OWASP. *OWASP Top Ten*. 2021. URL: https://owasp.org/www-project-top-ten/.

Appendices

Table of Contents

A Grafana

Figure 3: *A snapshot of the Grafana dashboard.*

B Sonarcloud

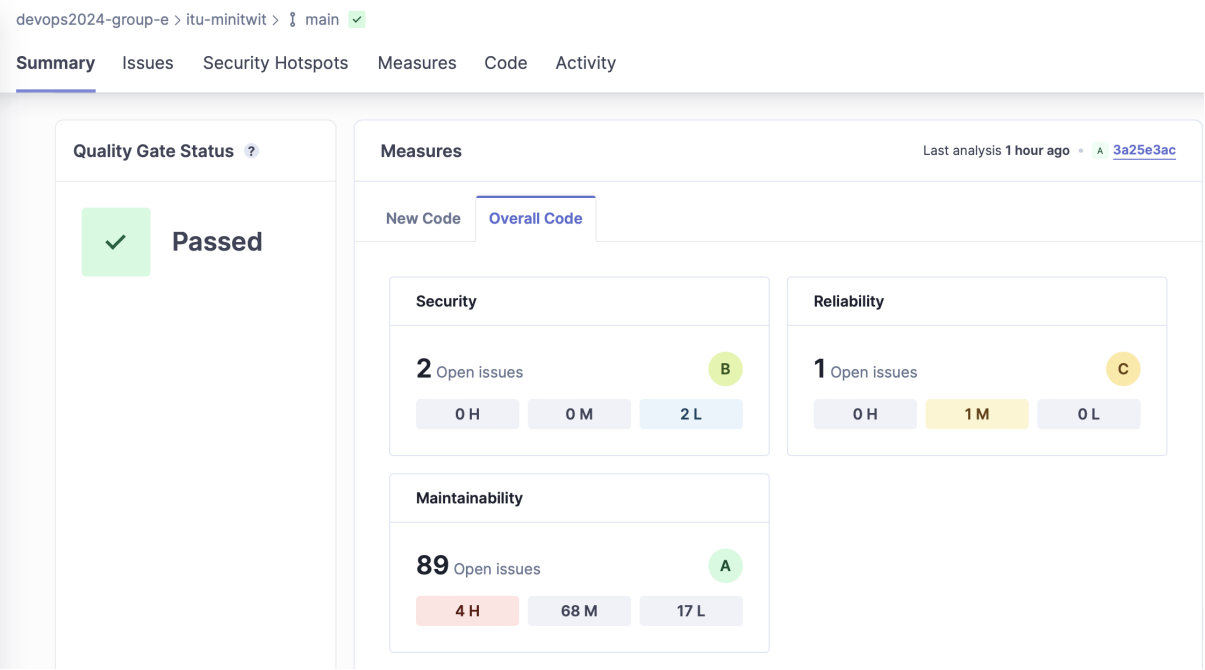

Figure 4: *Sonarcloud code analysis.*

C Security Assessment

C.1 Risk Identification

C.1.1 Asset Identification

By browsing through our setup and documentation, we identified the following list of assets:

- Web application
- Public GitHub repository
- Digital Ocean servers
- Tools
  - Ansible
  - Code Climate
  - Docker, Docker Compose & Docker Swarm
  - GitHub Actions
  - Grafana
  - Linters
  - Loki
  - Pulumi
  - Sonar Cloud

C.1.2 Threat Source Identification

To help us identify possible threats to the system, we have consulted the OWASP *Top 10 Web Application Security Risks*[3], which describes the following threats:

1. Broken Access Control
2. Cryptographic failures
3. Injection attacks
4. Insecure Design
5. Security misconfiguration
6. Vulnerable and outdated components
7. Identification and authentication failures
8. Software and Data integrity failures

9. Security Logging and Monitoring Failures
10. Server Side Request Forgery

C.1.3 Risk Scenario Construction

Based on the information gathered from the two previous steps, we have constructed the following risk scenarios and outlined which of the OWASP top 10 risks affecting the scenario:

1. **URL Tampering**

   The attacker can construct URL in the **web application** with user ID such that they bypass login and are able to write message from another user's account.

   This would be an issue of *Broken Access Control* and *Server Side Request Forgery*.

2. **Log Injection**

   The attacker can fabricate log information via injection attack in the **web application** as a means to hide their activity and ill-intentioned actions.

   This would be an issue of *Security Logging and Monitoring Failures* and *Injection attacks*.

3. **Password Brute Forcing**

   The attacker can brute force login credentials in the **web application** by taking advantage of no timeouts and weaker hash implementation.

   This would be an issue of *Cryptographic failures* and *Identification and authentication failures*.

4. **Depricated Dependencies**

   The attacker can identify weak, outdated or depricated tools and dependencies in the system's CI/CD pipeline via **GitHub Actions**, which is publicly availble through the **GitHub repository**.

   This would be an issue of *Vulnerable and outdated components* and *Software and Data integrity failures*.

5. **Open Ports**

   The attacker can scan the public IP addresses of the **Digital Ocean** servers to find unnecessarily or unexpectedly open ports with known vulnerabilities, which can be exploited in further attacks.

   This would be an issue of *Security misconfiguration*.

6. **SQL Injection**

   The attacker can target the login page of the **web application** with SQL-injection attacks to strike the database.

   This would be an issue of *Injection attacks*.

7. **Exposed Secrets**

   The attacker can get access to the system or associated tools via secrets found written in the code in the public **GitHub repository**.

   This would be an issue of *Identification and authentication failures* and *Security misconfiguration*.

C.2 Risk Analysis

C.2.1 Likelihood Analysis

Likelihooed will be graded on the following scale:{Rare, Unlikely, Possible, Likely, Certain} with *Rare* being the least severe and *Certain* being the most severe.

1. **URL Tampering**

   We examined the different URLs on the web application and couldn't find any where the ID's or login parameters were exposed. Therefore we deemed the likelihood to be **Unlikely**.

2. **Log Injection**

   We got warnings from Sonar Cloud that we had logging vulnerabilities in our code several different places. Therefore we deemed the likelihood to be **Likely**.

3. **Password Brute Forcing**

   We currently don't have any measures in place to combat brute force attacks and no password requirements for the users when signing up. Therefore we deemed the likelihood to be **Possible**.

4. **Depricated Dependencies**

   Our GitHub repository and thereby our workflows for GitHub Action as well are public for anyone to see. We are in the workflows using templates of actions made available and written by other alongside showcasing some of various tools we use. We currently rely on the fact that the actions and tools we use are secure, but haven't incoporated anything that checks whether that is true. However, many of the tools we use are well-known and therefore we hope that known vulnerabilities are getting discovered rather quickly. Therefore we deemed the likelihood to be **Possible**.

5. **Open Ports**

   All of the IP addresses for our servers are public in Digital Ocean, which makes it very easy to scan for vulnerabilities. We have put up firewalls and have taken measures to only have necessary ports open, but are also aware that accidental port exposure, through some of the tools we are using, is possible. We are convinced that this is how our server got hacked during the course. Therefore we deemed the likelihood to be **Certain**.

6. **SQL Injection**

   We have made sure to sanitize user input on the login page of the web application. Therefore we deemed the likelihood to be **Rare**.

7. **Exposed Secrets**

   We have been conscious to ensure that secrets were either kept locally where only ourselves can access them such as secret keys for logging into the servers or used the 'environment secrets' tool on GitHub if secrets had to be accessed from the repository. Additionally, we have not made generic passwords, but rather used random password generators to get stronger passwords. Therefore we deemed the likelihood to be **Rare**.

C.2.2 Impact Analysis

Impact will be graded on the following scale:{Insignificant, Negligible, Marginal, Critical, Catastrophic} with *Insignificant* being the least severe and *Catastrophic* being the most severe.

1. **URL Tampering**

   This would breach both the confidentiality and the integrity of the system's security. We still keep all our data, though the data would have been compromised. Therefore we deemed the impact to be **Critical**.

2. **Log Injection**

   This could be used to disguise an attackers activity and attack attempts on the web application. However, it wouldn't give them access to the server or applicition itself nor other data than the logs. We would still want to know if someone was trying to attack our system. Therefore we deemed the impact to be **Marginal**.

3. **Password Brute Forcing**

   This would breach both confidentiality and integrity. In very severe cases, it could also affect the availability, if the requests to login became to intense. We would still keep all of our data, though the data would have been compromised. Therefore we deemed the impact to be **Critical**.

4. **Depricated Dependencies**

   This would have a very big attack surface, as we most likely wouldn't know which tool or where in the application process we could have a vulnerability. The target for a vulnerability could thereby vary in severity, but could in the worst case have severe consequences. Therefore we deemed the impact to be **Catastrophic**.

5. **Open Ports**

   This again would have a big attack surface, as we don't know which tool or where in the application we potentially could have a vulnerability exposing ports for an adversary to attack. If the attacker ended up gaining access to our servers, they would have full

control over the application in the worst case. Therefore we deemed the impact to be **Catastrophic**.

6. **SQL Injection**

   This would breach both confidentiality and integrity. The data could both be compromised and lost, but we would have a backup of the database. Therefore we deemed the impact to be **Critical**.

7. **Exposed Secrets**

   If any of the secret in the GitHub repository were to fall into an attackers hands, they would be able to get access to our setup and thereby dismantle the entire system. Therefore we deemed the impact to be **Catastrophic**.

C.2.3 Risk Matrix

Based on the points made about the likelihood and impact of each risk scenario, we have constructed the following risk matrix to indicate the severities and prioritize the scenarios:

	Rare	Unlikely	Possible	Likely	Certain
Catastrophic	Exposed Secrets		Depricated Dependencies		Open Ports
Critical	SQL Injection	URL tampering	Password Brute Forcing		
Marginal				Log Injection	
Negligible					
Insignificant					

Figure 5: *Risk matrix based on security assessment*

C.2.4 Action Plan

We discussed the results of the security assessment and decided on focusing on the risk scenarios placed in the red area of the risk matrix.

For the open ports, we went through our firewalls setting and scanned our main server's IP address to see the current open ports. We tried to close the port we had open for Prometheus, but got conflicting results, when we checked the firewall istelf compared to the scan of the IP address.

For the log injection, we went through our code and ensured that all user data was sanitized before added to the log, such that it couldn't be compromised with.

For the depricated dependencies, we added the tool *dependabot* to our CI/CD pipeline, which makes sure that the dependencies used in our system are up to date and thereby less vulnerable to older known exploits.

For the password brute forcing, we decided to leave it as it, due to the group not having enough information on how the creation of users and login works in the simulator and thereby not knowing if the simulator could handle password restraints or 2-factor authentication, which would have been our solution to improve this issue.